Scaling Natural-Language Graph-Based Test Time Compute for Automated Theorem Proving

Vincent Li * 1 Tim Knappe * 2 Yule Fu 3 Kevin Zhu 4 Kevin Han 4

Abstract

Large language models have demonstrated remarkable capabilities in natural language processing tasks requiring multi-step logical reasoning capabilities, such as automated theorem proving. However, challenges persist within theorem proving, such as the identification of key mathematical concepts, understanding their interrelationships, and formalizing proofs correctly within natural language. We present KG-prover, a novel framework that leverages knowledge graphs mined from reputable mathematical texts to augment general-purpose LLMs to construct and formalize mathematical proofs. We also study the effects of scaling graph-based, test-time compute using KG-Prover, demonstrating significant performance improvements over baselines across multiple datasets. General-purpose LLMs improve up to 21% on miniF2F-test when combined with KG-Prover, with consistent improvements ranging from 2-11% on the ProofNet, miniF2F-test, and MUSTARD datasets without additional scaling. Furthermore, KG-Prover with o4-mini achieves over 50% miniF2F-test. This work provides a promising approach for augmenting natural language proof reasoning with knowledge graphs without the need for additional finetuning.

1. Introduction

The advent of Large Language Models has revolutionized natural language processing, enabling machines to perform complex reasoning tasks using transformer models (Vaswani et al., 2023; Peters et al., 2018; Brown et al., 2020; Srivastava et al., 2023). Transformer-based models have shown promise in mathematical problem-solving, which inherently requires multi-step logical inference and a precise understanding of abstract concepts (Robinson and Voronkov, 2001; Guo et al., 2025). Despite these advancements, significant challenges remain in automating the identification of mathematical concepts, understanding their interrelations, and formalizing proofs within a mathematical framework (Hendrycks et al., 2021). Work by (Polu and Sutskever, 2020) introduced training language models to generate proofs in formal languages and use such models to address the generation of original mathematical terms – leading to the introduction of the GPT-f proof assistant for the Metamath formalization language. Systems such as InternLM2.5-StepProver and DeepSeek-Prover-V2 directly generate proof candidates in the Lean language, achieving state-of-the-art performance in a wide variety of theorem proving benchmarks (Wu et al., 2024; Ren et al., 2025).

Recent advances in AI-driven mathematics have targeted the integration of neurosymbolic architectures with formal verification frameworks. Systems such as DeepMath and HOList employ MCTS guided by graph neural networks to prune combinatorial proof spaces (Bansal et al., 2019; Alemi et al., 2017). These frameworks combine self-play reinforcement learning with backward-chaining, enabling exploration of lemma sequences in interactive theorem provers.

A parallel line of research explores the use of natural language as an intermediate representation for guiding formal reasoning. Notably, Jiang et al. (2023) introduced a draft-sketch-prove pipeline, in which informal proof sketches are first generated in natural language and then incrementally translated into formal code. This enables the model to exploit the flexibility of natural reasoning, though at the cost of potential errors and ambiguity during the translation into a formal theorem proving language such as Lean. Theorem-Llama attempts to bridge the gap between natural language (NL) reasoning and formal language (FL) proofs using an NL-FL aligned dataset for training while still integrating NL text in the proof (Wang et al., 2024).

In this work, we introduce KG-Prover, a novel automated theorem-proving framework allowing a general purpose LLM to semantically retrieve and traverse a knowledge

The second AI for MATH Workshop at the 42nd International Conference on Machine Learning, Vancouver, Canada *Equal contribution [1]Boston University [2]Provadis School of International Management & Technology [3]Duke University [4]Algoverse AI Research. Correspondence to: Vincent Li <vinli@bu.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

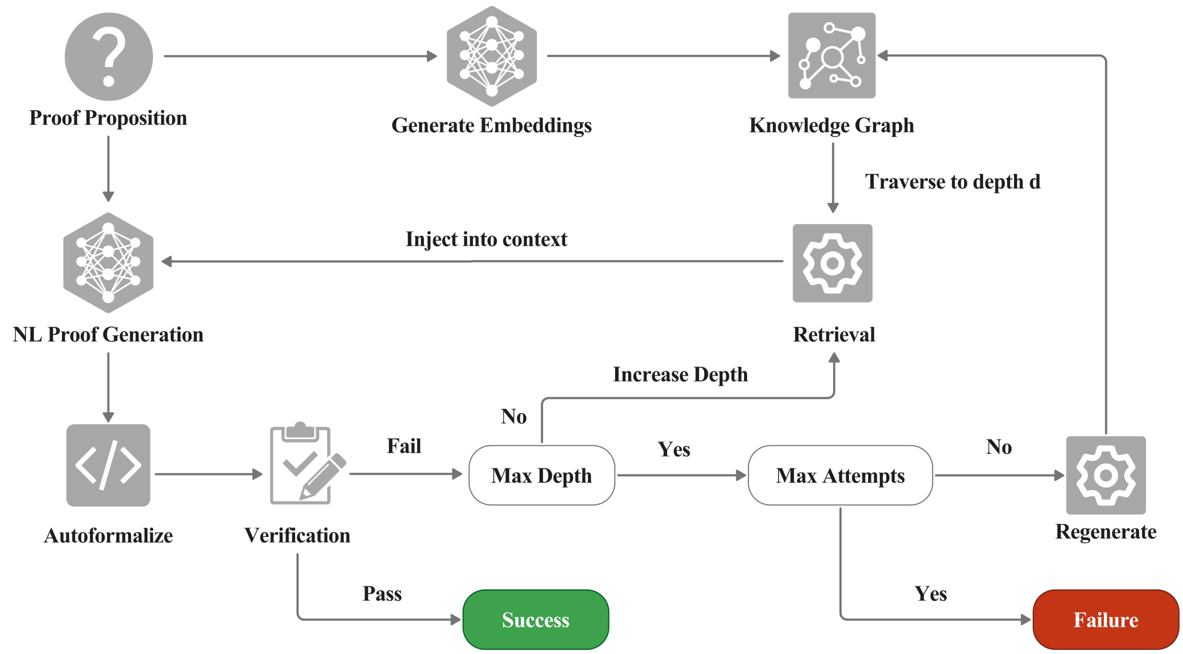

Figure 1. Whereas many modern proof systems focus on training time improvements, we integrate Node retrieval based on an interconnected knowledge graph into our proof system at inference time. Before generating a proof, we inject the most similar nodes into the context, then verify the proof using Lean. If the verification is unsuccessful, we grant the model the chance to traverse the graph deeper, where the knowledge graph allows it to explore other related concepts and theorems, on multiple attempts.

graph derived from ProofWiki.

KG-Prover begins with selecting a starting node via semantic embedding lookup, followed by iteratively and selectively expanding the traversal of covered nodes by a judging process. The general-purpose LLM then generates an informal proof which is then autoformalized and verified using Lean (de Moura and Ullrich, 2021; Zhu et al., 2025). We show that effective test time compute scaling is achievable by simply modulating the graph traversal depth.

Unlike previous approaches, our framework does not rely on large amounts of formal training data or intensive expert iteration. Instead, we operate with no specialized training, leveraging the built-in natural language reasoning abilities of general-purpose LLMs to synthesize graph-retrieved information before outputting an informal proof.

Our contributions are as follows:

- Develop KG-Prover, an automated theorem proving framework relying on natural language informal proof generation combined with an iterative refinement-based knowledge graph traversal and an LLM as a judge.
- We build a knowledge graph using ProofWiki of over 60,000 nodes and 300,000 edges that represent mathematical concepts and their interrelations, modeling complex relationships with mathematically similar subjects.
- We introduce an iterative refinement system based on a heuristic evaluation by a model judge and beam search for further revisions, improving performance by up to 26.4% over baseline and 21.8% over the non-scaling KG-Prover.

2. Related Work

Learning-Based Formal Provers Recent advancements in theorem proving have increasingly focused on integrating structured knowledge with LLMs. Notably, DeepSeek-Prover-V1.5 (Xin et al., 2024) combines reinforcement learning from proof assistant feedback (RLPAF) with Monte-Carlo tree search. The model, pre-trained on formal mathematical languages like Lean 4, achieves state-of-the-art results on miniF2F-test and ProofNet. It does so by dynamically exploring diverse proof paths through intrinsic-reward-driven search. This builds on earlier work such as LeanDojo (Yang et al., 2023), which introduced ReProver, an LLM-based prover enhanced with retrieval capabilities to efficiently select theorem premises. Similarly, HyperTree Proof Search (Polu and Sutskever, 2020) demonstrated that structured search algorithms could enhance proof generation in formal systems like Metamath. Furthermore, Wu et al. (2022) showed that LLMs can effectively translate informal mathematical statements into formal logic, targeting Isabelle/HOL, proving that the resulting autoformalized specifications are sufficiently accurate to improve downstream formal provers trained on them, which shows increasing capabilities with scaling steps such as stepwise iterative refinement(Zhang et al., 2024).

Sampling and Compute Strategies Additionally, (Hübotter et al., 2024) proposes a "compute-optimal" strategy that dynamically adjusts resources based on task difficulty. This approach achieves efficiency gains over traditional sampling and allows smaller models to outperform larger counterparts in FLOPs-matched evaluations. The strategy is broadly applicable in various complex reasoning domains, including automated theorem proving.

Feedback Mechanisms and Self-Improving Agents In improving feedback mechanisms, STP (Dong and Ma, 2025) uses self-play between conjecturer and prover agents, while Formal Theorem Proving by Hierarchical Decomposition (Dong et al., 2024) rewards lemma decomposition via reinforcement learning. Finally, the MUSTARD project (Johnson et al., 2020) used an iterative approach where the LLM generates a problem, constructs an informal proof, converts it into Lean (de Moura et al., 2015) format, and verifies the proof with a Lean interpreter, while its concurrent framework addresses mathematical language grounding via structured semantic parsing (Johnson et al., 2020). MUSTARD operates in three stages: sampling concepts, using generative models to create problems and solutions, and employing proof assistants to validate these solutions. Jiang et al. (2023) proposed a three-phase framework that first drafts an informal proof in natural language, then sketches a rough tactic script in Lean, and finally invokes a formal prover to complete the remaining subgoals. This approach illustrates how guidance can yield strong formal results, supporting our use of informal proof generation as a first-class component.

Graph LMs and Retrieval Mechanisms Graph-based retrieval-augmented generation techniques have also received growing attention for their ability to leverage structured relationships to enhance downstream tasks such as question answering and formal proof search. For instance, GraphRetriever combines a graph-structured knowledge base with question embeddings to systematically identify salient nodes for more focused generative reasoning, outperforming text-only retrieval systems in factual QA tasks (Wang et al., 2022). Similarly, QAGNN introduces a graph neural network that encodes question-relevant knowledge subgraphs, thereby enabling more interpretable and accurate reasoning within language model generation (Verma et al., 2023). Beyond question answering, hybrid systems like GraFormer exploit graph-based encoders to refine contextual embeddings retrieved from large corpora, demonstrating improved performance in specialized domains (Zhao et al., 2021). Collectively, these works underscore the capabilities of knowledge graphs with LLMs for reasoning tasks, providing more effective retrieval.

Lean Provers Recent works in direct Lean proving have shown promising advances in consistently formalizing correct and rigorous mathematical proofs. By training and fine-tuning LMs such as InternLM2.5-StepProver and DeepSeek-Prover-V2 to generate directly in Lean's formal language, these systems demonstrate state-of-the-art performance in autoformalization tasks (Wu et al., 2024; Ren et al., 2025). InternLM 2.5–StepProver applies expert iteration entirely within Lean, using curriculum learning and self-generated proofs to continually improve a fine-tuned policy model. In parallel, DeepSeek-Prover V2 leverages a large language model to recursively decompose theorems into subgoals, combining this with reinforcement learning shaped by verifier feedback. Both approaches treat the Lean environment as an interactive medium and fully disregard natural language during inference.

Earlier efforts, such as TheoremLlama (Wang et al., 2024), demonstrated that even mid-sized open models can reach strong formal proving performance when trained on bootstrapped Lean–natural language pairs.

Integrating Graphs and forming proofs in natural language Our model extends prior work by integrating a ProofWiki-derived knowledge graph with large language models for automated proof generation. Using natural language as an intermediate representation allows access to a much broader corpus of LaTeX-based and informal proofs than formal codebases like Lean. It also harnesses LLMs' emergent reasoning abilities and exposes interpretable reasoning traces that can reveal novel strategies. The tradeoff is added error and complexity in the informal-to-formal translation, especially in semantically precise edge cases.

We address this with a two-agent system for informal proof generation and formalization, supported by retrieval-augmented generation over graph-structured knowledge. This follows trends in autoformalization seen in DeepSeek-Prover-V1.5, which combines RL and tree search, and TheoremLlama, which shows gains from natural language intermediaries. Our graph-based retrieval also aligns with work like GraphRetriever and QAGNN, where structure enables targeted, interpretable context. Iterative refinement and verification loops reflect recent advances in dynamic test-time compute (Hübotter et al., 2024). Together, these elements advance scalable, interpretable, modular theorem proving with LLMs.

3. Methodology

Our framework automates mathematical proof generation by integrating LLMs with a knowledge graph constructed from ProofWiki. We employ a multi-stage approach combining retrieval-augmented generation with a multi-step LLM system for proof formalization. The system consists of four main components: 1) knowledge graph traversal, 2) informal proof generation, 3) formal proof generation, 4) verification and refinement. Figure 1 illustrates the overall KG-Prover workflow.

3.1. Knowledge Graph Construction

We built the underlying KG-Prover knowledge graph by parsing ProofWiki, an online compendium organized into distinct namespaces such as "Definition," "Axiom," and "Proof." By targeting these namespaces, we reliably extracted the formal components: the precise definition statements, axiom listings, theorem propositions (including lemmas and corollaries), and their corresponding proof details (ProofWiki, 2025). In our pipeline, each node represents a self-contained mathematical assertion–such as the text of a definition, the formal proposition of a theorem or lemma, or a corollary–while the textual proof that follows is stored as a property or linked entity. Hyperlinks within pages (e.g. references to earlier theorems or definitions) become edges in the graph, capturing higher level dependencies and conceptual relationships. We store this graph in a Neo4j (Webber, 2012) graph database, augmenting each node with precomputed embedding vectors using OpenAI's text-embedding-3-large model. An example entry from our nodes collection can be found in Appendix E.1. This structure enables efficient semantic queries: given a problem statement, we compute its embedding and retrieve the top-k most similar nodes to provide as a starting point for subsequent graph traversals. Explicit details on the construction of the knowledge graph can be found in Appendix F.

3.2. KG-Prover

Our KG-Prover framework consists of four main components: 1) knowledge graph traversal, 2) informal proof generation, 3) formal proof generation, 4) verification and refinement.

3.2.1. Retrieval

Let $G = (V, E)$ be a knowledge graph, where V represents all nodes as mathematical theorems and E represents the edges between them. Given a proposition P that we are tasked to prove, we use the below-signified similarity function that assigns a relevance score to each node based on its similarity to P.

We opt for cosine similarity by generating an embedding vector for P, $\mathbf{v}_P$, and comparing $\mathbf{v}_P$ to the other node embeddings $\mathbf{v}_i \in V$ in the knowledge graph:

$$S = sim(v_P, v_i) = \frac{v_P \cdot v_i}{\|v_P\|_2 \|v_i\|_2}$$

If P is not solved in the first iteration of generation, we introduce a depth parameter d that can be incremented up to an allowed depth D. We iteratively expand the context by selecting up to k additional nodes that are related concepts of previously selected nodes.

$$k_1, k_2, \ldots, k_i = \arg \max_{V_{d-1} \in V} S(V_d, V_{d-1})$$

where V_{d-1} represents the set of all traversed nodes and V_d represents the set of all 1-hop neighbors of V_{d-1}. Mathematically V_{d-1} is defined as the set of $v_j : \exists (a, b) \in E$ where $a \in V_{d-1}$. For each traversal, we select the top k 1-hop neighbors ranked by our similarity function S.

This expansion continues until either:

- P is resolved by the language model.
- The maximum depth D is reached and the amount of regenerating tries is expended.

3.2.2. Informal Proof Generation

Informal proof generation integrates retrieved-context into the language model prompt and uses the LLM to create an informal, natural language proof based on this enhanced input. Initially, generating a natural language proof allows KG-Prover to take advantage of the repository of known proofs outlined within the knowledge graph. If the proof fails to pass the verification stage, the framework iteratively deepens the context by one level in the knowledge graph, selecting the top-k semantically closest neighboring nodes to uncover missing key concepts. The updated context is then used for subsequent proof generation attempts.

All generator models are invoked with a 32 k-token context window (8 k for LLaMA 3 8B). When the problem statement plus retrieved snippets exceed the window we (i) keep the full problem, (ii) retain at most the first N_s=4 sentences of each snippet, and (iii) drop the least-similar node(s) until the prompt fits. This "similarity $\rightarrow$ sentence $\rightarrow$ node" back-off preserves high-ranked information while guaranteeing admissible length.

3.2.3. Formal Proof Generation

The autoformalization LLM ingests the code prefix, proposition, and informal proof before generating the translated formal proof. The framework for both the Autoformalizer and the generator models can be found in Appendix E.2.1. Finally, the autoformalizers' output is parsed to extract well-specified Lean 4 code.

3.2.4. Verification and Refinement

To ensure the formal correctness of the proofs generated by our framework, we adopted the Lean verification method from DeepSeek-Prover-V1.5 to enhance the formalization step in our proof generation process, utilizing RLPAF to refine our model's ability to generate proofs that are verifiable in Lean (Jiang et al., 2024). By integrating proof-assistant feedback, our models are more robust in producing proofs that adhere to the strict syntactic and logical requirements of Lean.

The formal proofs were verified using Lean 4 to ensure correctness. The generated proof code was submitted to Lean, and the results were analyzed. On failure, we extract error messages and feed them back into the autoformalizer along with adjusted prompts. This loop repeats until verification succeeds or we exhaust the allowed attempts r. If the maximum depth is not yet reached, we perform another graph traversal and generation loop.

3.3. Scaling KG-Prover

Self-consistency has proven itself as strongly effective, on commonly used reasoning as well as mathematical tasks, making use of the different approaches a language model might take while sampling multiple responses (Wang et al., 2023). To utilize this phenomenon and improve robustness and accuracy under limited graph traversals, we incorporate test-time scaling, sampling, and search strategies:

3.3.1. Best-of-N

For each proof task, we generate n independent informal proofs, autoformalize, and verify them. A dedicated model then acts as a judge, evaluating each candidate's proof across dimensions of mathematical correctness, clarity, and reasoning completeness. The judge assigns scores from 0-10 and provides justification for each evaluation. Candidates are then sorted by their scores, with the highest-scoring proof selected as the "optimal" solution to convert into Lean.

3.3.2. Beam Search

We organize proof candidates into a beam of width w and search depth s. At each step, the top-w candidates are expanded by generating refinements based on verifier feedback (Sun et al., 2023). These refinements are then scored and ranked, with the top-k candidates retained for subsequent iterations. The process repeats s times, ultimately returning the "best" proof that is both high-quality in terms of interpretability and formally verifiable. This balances the exploration of diverse proof paths with verification-driven refinement.

4. Experiment Design

4.1. Models

To create semantic representations in the form of embeddings, we used OpenAI's `text-embedding-3-large` model (Neelakantan et al., 2022).

For informal proof generation, we utilized GPT-4o-mini, as well as Claude 3.5 Sonnet and a collection of LLAMA 3 models (OpenAI, 2024; Anthropic, 2024; AI, 2024). We measure performance on the COT-reasoning models Deepseek-R1, o1-mini, and o4-mini (Team, 2025).

As an Autoformalizer we use DeepSeek-Prover-V1.5 (Jiang et al., 2024), which is an open-source language model, designed for theorem proving in Lean (de Moura and Ullrich, 2021). We use the model explicitly only for the translation of the already generated informal proof into Lean format to validate informal proofs.

4.2. Datasets

To evaluate the effectiveness of our framework, we conducted experiments on multiple benchmarks commonly used in automated theorem proving: **miniF2F**[1], **ProofNet**, and **MUSTARDSAUCE** (Zheng et al., 2022; Azerbayev et al., 2023; Huang et al., 2024). MiniF2F is a benchmark dataset of formal mathematics problems sourced from undergraduate-level mathematics competitions. ProofNet is a large-scale dataset of mathematical proofs and theorem statements, ranging in difficulty and domain. MUSTARDSAUCE is the dataset MUSTARD generated itself using GPT-4.

Our exact dataset configuration can be found in Appendix G.

4.3. Baselines

We compare out system to a pure zero-shot chain-of-thought prompt with three verification retries (r=3) and a naive *RAG* approach that adds the top-5 embedding-retrieved snippets but keeps the same single-completion, single-retry budget.

4.3.1. Retrieval Augmented Generation

The *RAG* rows in our tables correspond to a lightweight retrieval baseline that injects a fixed natural-language context but *does not* apply any graph traversal or extra test-time compute.

(i) Dataset and corpus. We retrieve exclusively from the same ProofWiki dump described in §3.1; no additional corpora are introduced.

[1]Unless stated otherwise, references to miniF2F denote the average performance across only the test split.

Dataset (↑)	Method	Claude 3.5 Sonnet	Deepseek R1	Llama 3.1 8B	Llama 3.3 70B	GPT 4o	o1 -mini
ProofNet	Base	2.69%	2.69%	3.76%	2.15%	3.23%	3.76%
	RAG	3.76%	3.76%	3.76%	3.76%	5.38%	5.91%
	KG-Prover	4.84%	5.38%	4.30%	4.30%	6.45%	**6.99%**
miniF2F	Base	22.95%	20.08%	20.49%	25.00%	23.36%	23.77%
	RAG	28.69%	22.54%	24.59%	24.59%	28.69%	28.28%
	KG-Prover	31.15%	28.28%	**31.97%**	30.74%	30.74%	30.74%
MUSTARD	Base	28.00%	20.00%	24.00%	25.60%	28.00%	24.80%
	RAG	28.40%	25.00%	28.00%	28.8%	28.00%	26.80%
	KG-Prover	30.00%	27.00%	27.60%	32.5%	30.00%	**34.00%**

Table 1. Results on ProofNet, miniF2F and MUSTARDSAUCE. *Base* and *RAG* generate a *single* completion per Lean retry (up to $r=3$); KG-PROVER may additionally apply beam search as detailed in §5.4. Values are the percentage of theorems proved; bold marks the largest gain over the corresponding Base.

(ii) Context selection method. For every problem we encode the statement with `text_embedding_3_large`, compute cosine similarity against all ProofWiki node embeddings, and prepend the titles and statements of the top–$k=5$ nodes to the prompt. We deliberately omit their proofs and neighbours to keep the context size small.

(iii) Test-time budget. We ask the generator model for **one** completion (temperature 0). The completion is autoformalised and verified in Lean; on failure we repeat the *same single-completion pipeline* up to $r=3$ times, mirroring the *Base* system. No best-of-N, beam search, or deeper traversal is used, so the token cost is $r\,(T_{\rm p}+T_{\rm r})$.

This setup isolates the value of injecting a handful of semantically relevant statements, allowing a clean comparison against the full KG-PROVER which *does* scale test-time compute.

4.4. Introduced Scaling Parameters

In our evaluations we vary six parameters:k (top-k nodes selected by semantic similarity at the current depth), r (attempts allowed per proof), d (maximum traversal depth), n (candidates generated in a best-of-N scheme), w (beam width), and $search_depth$ (beam-search depth).

5. Results

5.1. Knowledge Graph Performance

As visualized in Table 1, using graphs consistently outperforms baseline proof systems and over Retrieval Augmented Generation. Performance gains of the KG-Prover ranged from 2-11% across different models[2]. Llama 3.1 8B achieved a 31.97% success rate on miniF2F, compared to a 20.49% baseline.

ProofNet represents the most challenging dataset with the lowest overall performance (2-7% success rates), attributed to the difficulty of the problems. They require higher abstract mathematical reasoning and more intricate proof structures. The miniF2F dataset showed moderate performance (20-31% success) because it includes more structured mathematical problems, intermediate complexity of proofs, and more predictable reasoning patterns.

MUSTARDSAUCE demonstrated moderate performance as well (24-34% success).

5.2. Finetuned foundation models

Model	minif2f
TheoremLlama (pass@128)	35.04%
TheoremLlama + KG-Prover	**36.89%**

Table 2. Using finetuned models with knowledge graph traversal depth = 6, witnesses improved performance over 128 rounds of generation

As visualized in Table 2, using structured Knowledge of our KG-Prover with a depth of 6, performs 1.85 percentage points better than a finetuned model outperforming the pass@128[3] on finetuned Lean provers.

[2]Although $top-k=5$ is a fixed parameter, the actual value can be smaller depending on the number of related nodes available at the current depth.

[3]We follow the definition of pass@k defined by Chen et al. (2021)

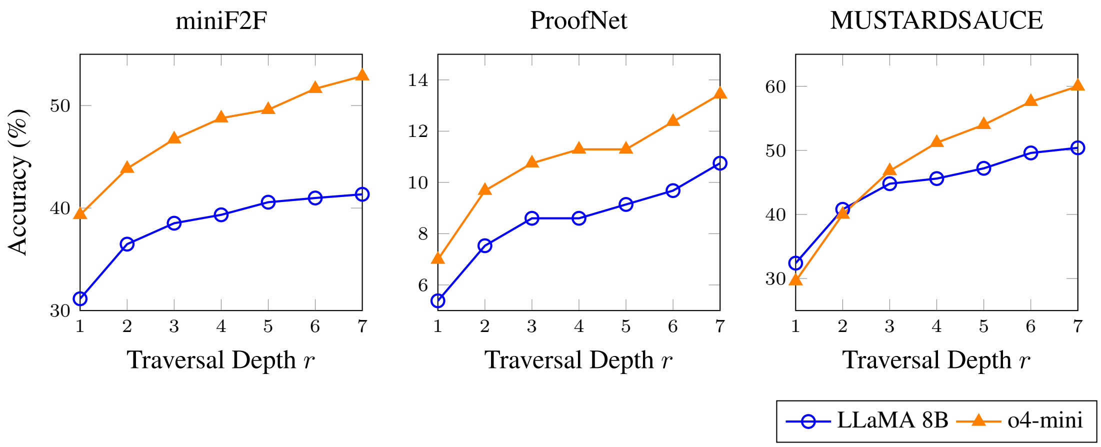

Figure 2. Comparing different depths for the beam search method on a set of parameters that are n = 5, beam width 3, search depth 2.

5.3. Scaling Traversal Depth

To allow the model for failure correction and improvement, the graph system has multiple consecutive attempts defined as r. Each attempt allows the model to traverse further in the graph and explore more nodes.

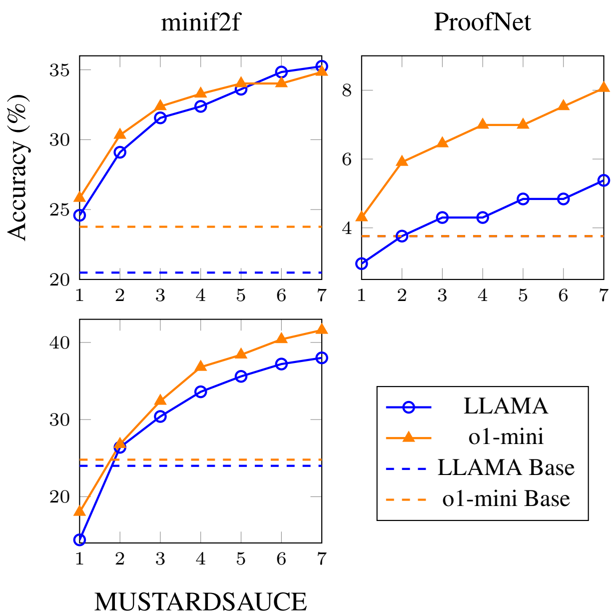

Figure 3. Accuracy increases with greater traversal depth r in the knowledge graph

As more proofs get injected into the context and the model gets more tries to correct initial mistakes, the accuracy scales higher per iterative refinement step. This effect is most predominant in smaller parameter models, such as Llama 3.1 8b. This behavior is captured in Figure 3. We can see that with more nodes injected, the performance rises.

5.4. Combined Scaling with Beam Search

As visualized in Figure 2, combining the knowledge graph with approaches that can sample responses from the context of the KG-Prover has a positive effect on accuracy across all benchmarks. Achieving up to 10.75% on ProofNet, 41.34% on miniF2F and 50.40% on MUSTARDSAUCE, which equates to improvements of 26.40% in the highest scaling configuration. Across all three dataset we find the first three depth increases to be the most effective in scaling the accuracy. While the leaps in accuracy flatten towards deeper depths, on harder datasets, we see the higher depths actually still bring a consistent improvement.

Additionally, we see that even on depth one, the performance consistently beats the baseline and on average performs on par[4] better than the KG-Prover without multiple candidate proofs.

For the larger context window of **o4-mini** the same configuration reaches 52.9% on miniF2F and 60.0% on MUSTARDSAUCE (Fig. 2), showing that KG-Prover scales favorably with model context (since it is a reasoning model).

5.5. Individual Cases

Qualitative inspection reveals clear prompt-and-response evolution across the scaling loop. Appendix I shows a miniF2F example: the zero-shot baseline fails, RAG retrieves a divisibility lemma and succeeds, KG-Prover depth-1 incorporates modular-arithmetic residues for a cleaner proof, and depth-3 adds an edge case that allows Lean to verify in one shot. Appendix J collects three failure cases (missing lemma, auto-formaliser mistranslation, and Lean timeout) to illustrate common error modes.

[4]considering slight deviations of $\pm$ 1%

Setting	# completions	Token budget	Cost multiplier
BASE	1	$T_\mathrm{p} + T_\mathrm{r}$	$1\times$
KG-PROVER	r	$r\,(T_\mathrm{p} + T_\mathrm{r})$	$r\times$
KG-PROVER + BEAM	$r\Big(n + w\sum_{i=0}^{s-1} w^{\,i}\Big)$	$r\Big(n + w\dfrac{w^s - 1}{w - 1}\Big)(T_\mathrm{p} + T_\mathrm{r})$	$r\Big(n + w\dfrac{w^s - 1}{w - 1}\Big)\times$

Table 3. Language-model usage per problem instance (r traversal attempts, initial beam of n candidates, beam width w, tree depth s).

5.6. Performance Tradeoffs

We isolate the cost that dominates monetary expenditure in practice: the number of language-model API calls and the prompt/response tokens for those calls consume. Autoformalization and Lean verification are performed on local hardware and are therefore ignored in this section.

Our general of usage of API calls per used method can be broken down into[5]:

- Base = 3 calls
- RAG = 3 calls
- KG-Prover = 3 calls
- KG-Prover + Beam (n=5,w=3,s=2) = 3×[2+2(3+1)] = 30 calls

TABLE NOTATION

Let T_p be the average prompt length (in tokens) for a single informal-proof request and T_r the average length of the LLM's response. Our methods differ only in how many *times* that request–response pair is issued.

IMPLICATIONS

- **Retries scale linearly.** Each additional attempt multiplies total spend by r but yields diminishing accuracy gains beyond r=3 (Figure 3).
- **Beam search is the price driver.** The geometric factor $n + w(w^s - 1)/(w - 1)$ explodes quickly: doubling the beam width from 3 to 6 would more than triple token usage while delivering $\leq$1 pp extra accuracy on PROOFNET.

6. Conclusion & Discussion

We present a framework that automates mathematical proof generation by integrating LLMs with a knowledge graph to utilize inter-dependencies across mathematical proofs. Our approach demonstrates the potential of combining multiple mathematical concepts in an intertwined graph. By doing so, language models can be effectively guided toward correct proof generation, resulting in improved accuracy and enhanced abilities in both reasoning through and formalizing proofs in natural language, whilst adding lean verification in a separate translation step.

We establish that existing foundation models can achieve similar or higher performing results as fine-tuned models, by simple context injections of related concepts during inference time, without requiring any additional pre-training, expert iteration, or training system of any kind. By doing this we witness performance increases across datasets of up to 11% by just using the KG-Prover and up to 26% when combined with proper scaling techniques.

7. Limitations

Despite the advancements in capturing semantic relationships in text via vectorized embeddings, embeddings can potentially suffer from issues such as loss of fine-grained logical structure and difficulties in preserving contextual dependencies across larger passages. This can lead to challenges in accurately retrieving relevant mathematical statements, especially in formalized settings where precise definitions and logical consistency are crucial. While we filter and discard irrelevant details, signs, and other minutiae, XML dumps can introduce noise that might disrupt of affect the semantic search and embeddings.

While our approach successfully formalizes proofs from structured datasets, its performance on entirely novel or highly abstract mathematical problems remains uncertain. Models trained on existing proofs may struggle with creative problem-solving or unconventional mathematical approaches.

Large Language Models have finite context windows, meaning lengthy or complex proofs may exceed the model's processing capacity. This might result in incomplete reasoning, loss of critical details, or forgetting earlier steps in multi-stage proofs.

Future work may enhance the knowledge graph and improve the autoformalization process to handle more complex mathematical concepts, improve language model translations or handling longer proofs using different attention mechanisms or.

[5]Under the standard configuration (r=3,k=5,d=3,n=1)

8. Reproducibility Statement

Our experiments were conducted using publicly available Datasets and Models. GPT-4o, 4o-mini, o1-mini and text-embedding-3-large can be accessed via https://openai.com/api/. Both Deepseek-R1 and the LLAMA 3 collection are open-sourced models. Claude models can be accessed via their respective API endpoints, under https://www.anthropic.com/api.

ProofNet and miniF2F, and MUSTARDSAUCE are publicly available datasets. Our Code is publicly available on GitHub; we encourage anyone to validate and extend our findings. The Neo4j-based graph database can be used under https://neo4j.com and could potentially be replaced with alternative graph databases as desired.

9. Ethical Considerations & Risks

Our knowledge base is derived from ProofWiki, an open database for formal proofs. While the page is moderated, adversaries could attempt to incorporate harmful content or incorrect factual information into the extracted pages. However, we consider this risk to be unlikely.

Although alignment work continues to progress Large Language Models can introduce biases towards certain marginalized groups or other minorities. All of our introduced models are moderated and have content filters that should prevent models from generating harmful content. However said filters aren't perfect, models can still be exploited via sophisticated prompting and other adversarial techniques. Given our contribution to the framework, we expect no increased risk in any of the given safety evaluation measures proposed.

9.1. GPU usage

GPU model	Watts	approx. usage Time
Nvidia A40	300 W	700 hours
Nvidia RTX A5000	300 W	50 hours

Table 4. Estimated GPU usage for all Evaluations.

The shown GPU usage may only partially reflect an accurate measure of the computational resources required, as major models are only available through API endpoints. We estimate the inference time on said APIs to be roughly 170 hours.

References

Meta AI. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Alex A. Alemi, Francois Chollet, Niklas Een, Geoffrey Irving, Christian Szegedy, and Josef Urban. Deepmath - deep sequence models for premise selection, 2017. URL https://arxiv.org/abs/1606.04442.

Anthropic. Claude 3.5 sonnet model card addendum, 2024. URL https://www.anthropic.com/news/claude-3-5-sonnet.

Z. Azerbayev, B. Piotrowski, H. Schoelkopf, E. W. Ayers, D. Radev, and J. Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2301.09469*, 2023.

K. Bansal, S. Loos, M. Rabe, C. Szegedy, and S. Wilcox. Holist: An environment for machine learning of higher-order theorem proving. In *International Conference on Machine Learning (ICML)*, 2019.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

L. de Moura and S. Ullrich. The lean 4 theorem prover and programming language. In *Proceedings of the In Automated Deduction – CADE 28: 28th International Conference on Automated Deduction*, 2021.

L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. The lean theorem prover (system description). In

A. Felty and A. Middeldorp, editors, *Automated Deduction – CADE-25*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, Cham, 2015.

Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving. *ICLR*, 2025.

Kefan Dong, Arvind Mahankali, and Tengyu Ma. Formal theorem proving by rewarding llms to decompose proofs hierarchically. *AAAI*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, and et al. Xiao Bi. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Yinya Huang, Xiaohan Lin, Zhengying Liu, Qingxing Cao, Huajian Xin, Haiming Wang, Zhenguo Li, Linqi Song, and Xiaodan Liang. Mustard: Mastering uniform synthesis of theorem and proof data, 2024. URL https://arxiv.org/abs/2402.08957.

Jonas Hübotter, Sascha Bongni, Ido Hakimi, and Andreas Krause. Efficiently learning at test-time: Active fine-tuning of llms. *arXiv preprint arXiv:2410.08020*, 2024.

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs, 2023. URL https://arxiv.org/abs/2210.12283.

X. Jiang, X. Hong, J. Wang, L. Wang, Q. Li, J. Guo, Z. Jin, and T. Zhao. Deepseek-prover-v1.5: Integrating reinforcement learning from proof assistant feedback for formal theorem proving. *arXiv preprint arXiv:2309.16443*, 2024.

A. Johnson, D. Baxley, A. Cheung, J. Halbrook, and M. L. Giger. Mathematics understanding through semantic theory and reasoning development (mustard). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13671–13672, 2020.

A. Neelakantan, T. Xu, R. Puri, A. Radford, J. M. Han, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022.

OpenAI. Gpt-4o mini: Advancing cost-efficient intelligence. OpenAI Blog, July 18 2024. Available at: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://aclanthology.org/N18-1202/.

S. Polu and I. Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

ProofWiki. Proofwiki: An online compendium of mathematical proofs, 2025. URL https://proofwiki.org. Accessed: 2025-01-31.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition, 2025. URL https://arxiv.org/abs/2504.21801.

Alan J. A. Robinson and Andrei Voronkov. *Handbook of Automated Reasoning, Volume 1*. 2001.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ambrose Slone, Ameet Rahane, Anantharaman S. Iyer, Anders Andreassen, Andrea Madotto, Andrea Santilli, Andreas Stuhlmüller, Andrew Dai, Andrew La, Andrew Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes, Arun Kirubarajan, Asher Mullokandov, Ashish Sabharwal, Austin Herrick, Avia Efrat, Aykut Erdem, Ayla Karakaş, B. Ryan Roberts, Bao Sheng Loe, Barret Zoph, Bartłomiej Bojanowski, Batuhan Özyurt, Behnam Hedayatnia, Behnam Neyshabur, Benjamin Inden, Benno

Stein, Berk Ekmekci, Bill Yuchen Lin, Blake Howald, Bryan Orinion, Cameron Diao, Cameron Dour, Catherine Stinson, Cedrick Argueta, César Ferri Ramírez, Chandan Singh, Charles Rathkopf, Chenlin Meng, Chitta Baral, Chiyu Wu, Chris Callison-Burch, Chris Waites, Christian Voigt, Christopher D. Manning, Christopher Potts, Cindy Ramirez, Clara E. Rivera, Clemencia Siro, Colin Raffel, Courtney Ashcraft, Cristina Garbacea, Damien Sileo, Dan Garrette, Dan Hendrycks, Dan Kilman, Dan Roth, Daniel Freeman, Daniel Khashabi, Daniel Levy, Daniel Moseguí González, Danielle Perszyk, Danny Hernandez, Danqi Chen, Daphne Ippolito, Dar Gilboa, David Dohan, David Drakard, David Jurgens, Debajyoti Datta, Deep Ganguli, Denis Emelin, Denis Kleyko, Deniz Yuret, Derek Chen, Derek Tam, Dieuwke Hupkes, Diganta Misra, Dilyar Buzan, Dimitri Coelho Mollo, Diyi Yang, Dong-Ho Lee, Dylan Schrader, Ekaterina Shutova, Ekin Dogus Cubuk, Elad Segal, Eleanor Hagerman, Elizabeth Barnes, Elizabeth Donoway, Ellie Pavlick, Emanuele Rodola, Emma Lam, Eric Chu, Eric Tang, Erkut Erdem, Ernie Chang, Ethan A. Chi, Ethan Dyer, Ethan Jerzak, Ethan Kim, Eunice Engefu Manyasi, Evgenii Zheltonozhskii, Fanyue Xia, Fatemeh Siar, Fernando Martínez-Plumed, Francesca Happé, Francois Chollet, Frieda Rong, Gaurav Mishra, Genta Indra Winata, Gerard de Melo, Germán Kruszewski, Giambattista Parascandolo, Giorgio Mariani, Gloria Wang, Gonzalo Jaimovitch-López, Gregor Betz, Guy Gur-Ari, Hana Galijasevic, Hannah Kim, Hannah Rashkin, Hannaneh Hajishirzi, Harsh Mehta, Hayden Bogar, Henry Shevlin, Hinrich Schütze, Hiromu Yakura, Hongming Zhang, Hugh Mee Wong, Ian Ng, Isaac Noble, Jaap Jumelet, Jack Geissinger, Jackson Kernion, Jacob Hilton, Jaehoon Lee, Jaime Fernández Fisac, James B. Simon, James Koppel, James Zheng, James Zou, Jan Kocoń, Jana Thompson, Janelle Wingfield, Jared Kaplan, Jarema Radom, Jascha Sohl-Dickstein, Jason Phang, Jason Wei, Jason Yosinski, Jekaterina Novikova, Jelle Bosscher, Jennifer Marsh, Jeremy Kim, Jeroen Taal, Jesse Engel, Jesujoba Alabi, Jiacheng Xu, Jiaming Song, Jillian Tang, Joan Waweru, John Burden, John Miller, John U. Balis, Jonathan Batchelder, Jonathan Berant, Jörg Frohberg, Jos Rozen, Jose Hernandez-Orallo, Joseph Boudeman, Joseph Guerr, Joseph Jones, Joshua B. Tenenbaum, Joshua S. Rule, Joyce Chua, Kamil Kanclerz, Karen Livescu, Karl Krauth, Karthik Gopalakrishnan, Katerina Ignatyeva, Katja Markert, Kaustubh D. Dhole, Kevin Gimpel, Kevin Omondi, Kory Mathewson, Kristen Chiafullo, Ksenia Shkaruta, Kumar Shridhar, Kyle McDonell, Kyle Richardson, Laria Reynolds, Leo Gao, Li Zhang, Liam Dugan, Lianhui Qin, Lidia Contreras-Ochando, Louis-Philippe Morency, Luca Moschella, Lucas Lam, Lucy Noble, Ludwig Schmidt, Luheng He, Luis Oliveros Colón, Luke Metz, Lütfi Kerem Şenel, Maarten Bosma, Maarten Sap, Maartje ter Hoeve, Maheen Farooqi, Manaal Faruqui, Mantas Mazeika, Marco Baturan, Marco Marelli, Marco Maru, Maria Jose Ramírez Quintana, Marie Tolkiehn, Mario Giulianelli, Martha Lewis, Martin Potthast, Matthew L. Leavitt, Matthias Hagen, Mátyás Schubert, Medina Orduna Baitemirova, Melody Arnaud, Melvin McElrath, Michael A. Yee, Michael Cohen, Michael Gu, Michael Ivanitskiy, Michael Starritt, Michael Strube, Michał Swędrowski, Michele Bevilacqua, Michihiro Yasunaga, Mihir Kale, Mike Cain, Mimee Xu, Mirac Suzgun, Mitch Walker, Mo Tiwari, Mohit Bansal, Moin Aminnaseri, Mor Geva, Mozhdeh Gheini, Mukund Varma T, Nanyun Peng, Nathan A. Chi, Nayeon Lee, Neta Gur-Ari Krakover, Nicholas Cameron, Nicholas Roberts, Nick Doiron, Nicole Martinez, Nikita Nangia, Niklas Deckers, Niklas Muennighoff, Nitish Shirish Keskar, Niveditha S. Iyer, Noah Constant, Noah Fiedel, Nuan Wen, Oliver Zhang, Omar Agha, Omar Elbaghdadi, Omer Levy, Owain Evans, Pablo Antonio Moreno Casares, Parth Doshi, Pascale Fung, Paul Pu Liang, Paul Vicol, Pegah Alipoormolabashi, Peiyuan Liao, Percy Liang, Peter Chang, Peter Eckersley, Phu Mon Htut, Pinyu Hwang, Piotr Miłkowski, Piyush Patil, Pouya Pezeshkpour, Priti Oli, Qiaozhu Mei, Qing Lyu, Qinlang Chen, Rabin Banjade, Rachel Etta Rudolph, Raefer Gabriel, Rahel Habacker, Ramon Risco, Raphaël Millière, Rhythm Garg, Richard Barnes, Rif A. Saurous, Riku Arakawa, Robbe Raymaekers, Robert Frank, Rohan Sikand, Roman Novak, Roman Sitelew, Ronan LeBras, Rosanne Liu, Rowan Jacobs, Rui Zhang, Ruslan Salakhutdinov, Ryan Chi, Ryan Lee, Ryan Stovall, Ryan Teehan, Rylan Yang, Sahib Singh, Saif M. Mohammad, Sajant Anand, Sam Dillavou, Sam Shleifer, Sam Wiseman, Samuel Gruetter, Samuel R. Bowman, Samuel S. Schoenholz, Sanghyun Han, Sanjeev Kwatra, Sarah A. Rous, Sarik Ghazarian, Sayan Ghosh, Sean Casey, Sebastian Bischoff, Sebastian Gehrmann, Sebastian Schuster, Sepideh Sadeghi, Shadi Hamdan, Sharon Zhou, Shashank Srivastava, Sherry Shi, Shikhar Singh, Shima Asaadi, Shixiang Shane Gu, Shubh Pachchigar, Shubham Toshniwal, Shyam Upadhyay, Shyamolima, Debnath, Siamak Shakeri, Simon Thormeyer, Simone Melzi, Siva Reddy, Sneha Priscilla Makini, Soo-Hwan Lee, Spencer Torene, Sriharsha Hatwar, Stanislas Dehaene, Stefan Divic, Stefano Ermon, Stella Biderman, Stephanie Lin, Stephen Prasad, Steven T. Piantadosi, Stuart M. Shieber, Summer Misherghi, Svetlana Kiritchenko, Swaroop Mishra, Tal Linzen, Tal Schuster, Tao Li, Tao Yu, Tariq Ali, Tatsu Hashimoto, Te-Lin Wu, Théo Desbordes, Theodore Rothschild, Thomas Phan, Tianle Wang, Tiberius Nkinyili, Timo Schick, Timofei Kornev, Titus Tunduny, Tobias Gerstenberg, Trenton Chang, Trishala Neeraj, Tushar Khot, Tyler Shultz, Uri Shaham, Vedant Misra, Vera Demberg, Victoria Nyamai, Vikas Raunak, Vinay Ramasesh, Vinay Uday Prabhu, Vishakh Padmakumar, Vivek Srikumar, William Fedus, William Saunders, William Zhang, Wout Vossen, Xiang Ren, Xiaoyu Tong, Xinran Zhao, Xinyi Wu, Xudong Shen, Yadollah Yaghoobzadeh, Yair Lakretz, Yangqiu Song, Yasaman Bahri, Yejin Choi, Yichi Yang, Yiding Hao, Yifu Chen, Yonatan Belinkov, Yu Hou, Yufang Hou, Yuntao Bai, Zachary Seid, Zhuoye Zhao, Zijian Wang, Zijie J. Wang, Zirui Wang, and Ziyi Wu. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2023. URL https://arxiv.org/abs/2206.04615.

Hao Sun, Xiao Liu, Yeyun Gong, Yan Zhang, Daxin Jiang, Linjun Yang, and Nan Duan. Allies: Prompting large language model with beam search. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3794–3805, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.247. URL https://aclanthology.org/2023.findings-emnlp.247/.

DeepSeek-AI Team. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Shreyas Verma, Manoj Parmar, Palash Choudhary, and Sanchita Porwal. Fusionmind – improving question and answering with external context fusion. *arXiv preprint arXiv:2401.00388*, 2023.

Dingmin Wang, Shengchao Liu, Hanchen Wang, Bernardo Cuenca Grau, Linfeng Song, Jian Tang, Song Le, and Qi Liu. An empirical study of retrieval-enhanced graph neural networks. *arXiv preprint arXiv:2206.00362*, 2022.

Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. Theoremllama: Transforming general-purpose llms into lean4 experts, 2024. URL https://arxiv.org/abs/2407.03203.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

J. Webber. A programmatic introduction to neo4j. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity (SPLASH '12)*, 2012.

Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models, 2022. URL https://arxiv.org/abs/2205.12615.

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. Internlm2.5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems, 2024. URL https://arxiv.org/abs/2410.15700.

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *NeurIPS*, 2023.

Lan Zhang, Xin Quan, and Andre Freitas. Consistent autoformalization for constructing mathematical libraries. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, page 4020–4033. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.233. URL http://dx.doi.org/10.18653/v1/2024.emnlp-main.233.

Weixi Zhao, Yunjie Tian, Qixiang Ye, Jianbin Jiao, and Weiqiang Wang. Graformer: Graph convolution transformer for 3d pose estimation. *arXiv preprint arXiv:2109.08364*, 2021.

K. Zheng, J. M. Han, and S. Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2022.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. Judgelm: Fine-tuned large language models are scalable judges, 2025. URL https://arxiv.org/abs/2310.17631.

A. Using Lean provers for informal proof generation

As discussed, our approach utilizes a two step process that generates an informal proof in natural language that is then translated and validated in Lean. As visualized in Table 5, even Lean provers can be enhanced using the KG-Prover that utilizes natural language.

Method	Theorem Llama	DeepSeek Prover-V1.5
minif2f Base	32.38	35.75
minif2f RAG	34.84	36.48
minif2f KG-Prover	**36.89**	**37.71**

Table 5. Lean-based provers show increased performance using the KG-Prover (traversal depth = 6) even when the Node data is in natural language.

This phenomenon demonstrates that even Lean-optimized and fine-tuned systems benefit from structured natural language knowledge encompassing related proofs.

B. Structural Improvement

Few-shot learning, even with briefly related examples, has been shown to improve performance across a variety of tasks and domains.

Therefore we hypothesize that even only partly related proof nodes will improve not only the proof understanding but will also benefit the structured formalization that is required for the correct interpretation and conversion of informal natural language into Lean4.

C. Judging the Best of N Tries

Interestingly, the results in Table 6 reveal a non-linear relationship in more challenging datasets like ProofNet, where an intermediate value (e.g., $N = 6$) did not always outperform a lower or higher N. This suggests that simply increasing the number of candidates is not universally beneficial; the quality of each candidate and the effectiveness of the judging mechanism play critical roles. As such, finding the right balance in model temperatures is crucial because an optimal setting enhances the judging process by providing a diverse pool of high-quality candidates[6].

D. Deterministic Evaluations

Unless specified otherwise, we use greedy decoding for all of our experiments. Additionally, the semantic search in our Graph knowledge base will yield identical outputs, given that the input doesn't change between different runs.

[6]Both best of N and best of N + tree search method evaluations had LLama 3.1 8B set on a temperature of 0.7

Dataset	Model	Best of N		
		N=2	N=6	N=10
ProofNet	Llama 8B	6.45%	5.38%	8.60%
miniF2F	Llama 8B	30.33%	30.74%	31.97%
Mustard	Llama 8B	30.00%	32.80%	33.6%

Table 6. Results by dataset *with the graph approach*, comparing "Best of N" values between 2 and 10.

While this behavior can be favorable in some situations, other evaluations may benefit from slight variations in different seeds. To introduce a slight stochasticity, other evaluations may vary the temperature parameter of the employed models, and use the introduced method in Appendix D.1 to introduce randomness into our knowledge graph.

D.1. Knowledge Graph Stochasticity

To mitigate fully repetitive outputs Nodes from the knowledge graph, we propose top-k shuffling, where we retrieve the k-highest ranked nodes, shuffle them, and select a subset. This method ensures diversity in individual generations. We favor this implementation over random sampling over a broader set of candidate nodes, selecting from a pool beyond the strict top-k. Due to the potentially less relevant knowledge, trading off precision for increased coverage.

The level of stochasticity can be tuned dynamically based on confidence scores or response variance metrics

E. Examples

E.1. Node example

- **from_id**: The ID of the current node.
- **to_id**: The ID of the linked node (found using the title-name-to-ID mapping).
- **type**: There are 6 different relationship categories:

```
USES_DEFINITION,
RELATED_DEFINITION,
USES_AXIOM,
SIMILAR_PROOF,
PROOF_DEPENDENCY,
PROOF_TECHNIQUE.
```

E.2. Prompt Examples

E.2.1. Prompt Example 1

The model was provided with the informal proof and a code template, and it generated the corresponding formal proof in Lean 4. Each element was processed to extract the title, namespace, and content.

```
You are a Lean 4 code generator.
We have:
HEADER:
{header}

INFORMAL PROOF:
{informal_proof}

PREFIX:
{informal_prefix}

STATEMENT:
{formal_statement}

GOAL (optional):
{goal}

INSTRUCTIONS:
1. Output exactly one triple-backtick
code block containing valid Lean 4 code.
2. Do not include any text or
explanations outside the code block.
3. Make sure it compiles in Lean 4.

Required Format:
# Start
```lean4
<Lean code here>
```
End
```

E.2.2. Prompt Example 2

```
You are a mathematics expert focused on
generating clear informal proofs.

Given the following mathematical
problem and context, generate a clear
and  detailed informal proof in
natural language.

Context: [Retrieved context]

Problem: [Problem statement]

Provide your proof in the following format:

Informal Proof:
[Your proof here]
```

F. Graph Dataset

We parsed an XML dump of ProofWiki, where each `<page>` element was processed. Irrelevant sections were filtered, and the wikitext was cleaned to obtain structured content.

F.1. Node structure

We represented each mathematical concept as a node in the knowledge graph, storing attributes such as:

- **id**: Unique identifier.
- **type**: Content type (e.g., definition, theorem).
- **title**: Page title.
- **name**: Extracted from the title.
- **content**: Theorems in algebraic notation.

G. Benchmarks

All datasets present their samples with natural language and a formal statement in Lean, which we use as ground truth to compare against.

By utilizing miniF2F, ProofNet, and MUSTARDSAUCE, we assess our framework's ability to generate and formalize proofs across diverse mathematical problems. The datasets provided a standardized evaluation setting, allowing us to compare our results uniformly with existing approaches and to analyze the strengths and limitations of our Method. However, it is possible that our setup deviates from the ones introduced in the respective papers of the dataset, which explains the varied performance across tasks, which is especially apparent on MUSTARDSAUCE. To set up a comparable evaluation, we compute the baseline of our setup as well, rather than taking the previous State-of-the-Art.

G.1. Used splits

We ran 186 problems from the test split of ProofNet, 244 problems from the test split of miniF2F, and randomly selected 250 theorem-proving problems from MUSTARDSAUCE.

H. Search Strategies within the Knowledge Graph

To optimize the process of automated proof generation, we explored different methods for navigating the constructed knowledge graph. Specifically, we implemented two primary search strategies: Breadth-First Search (BFS) and semantic search using vector embeddings. This section elaborates on these methodologies, their implementation in our framework, and analyzes their respective advantages and disadvantages in our scenario.

H.1. Breadth-First Search (BFS)

Breadth-First Search is a classic graph traversal algorithm that systematically explores the vertices of a graph in layers, starting from a given root node and expanding outward to neighboring nodes at increasing depths. In our framework, BFS was utilized as follows:

1. **Zero-Shot Prompting**: We initially present the problem statement directly to the GPT model without any additional context, requesting a proof in a zero-shot setting.
2. **First-Level Traversal**: If the zero-shot attempt is unsuccessful, we perform a BFS to explore the immediate neighboring nodes of the problem statement node. Specifically, we retrieve up to the nearest 50 nodes connected directly to the root node.
3. **Contextual Prompting**: We then prompt the GPT model again, providing the problem statement along with the content from the retrieved neighboring nodes to supply additional context for proof generation.
4. **Iterative Expansion**: If the proof remains incomplete or incorrect, we extend the BFS to the next level by including nodes that are two edges away from the root, effectively expanding the context window before re-prompting the GPT model.

The advantage of BFS is that it allows for a systematic exploration of the knowledge graph, ensuring that all nodes within a certain depth are considered, which may uncover relevant but non-obvious connections. By incrementally increasing the depth of traversal, we can control the amount of additional information provided to the GPT model, potentially improving the quality of the generated proof.

However, BFS can be computationally expensive, especially in densely connected graphs, as the number of nodes grows exponentially with each additional level of depth. Including a broad set of neighboring nodes may introduce irrelevant or redundant information, which could overwhelm the GPT model and hinder its ability to generate a coherent proof.

H.2. Semantic Search Using Embeddings

Semantic search leverages vector embeddings to identify nodes that are semantically similar to a given query (Neelakantan et al., 2022). Each node in our knowledge graph is associated with a high-dimensional embedding vector, enabling similarity computations.

Hierarchical Prompting: Similar to the BFS approach, we begin with a zero-shot prompt. If unsuccessful, we incrementally include the most similar nodes into the context when re-prompting the GPT model, effectively performing one-shot, two-shot prompting, and so on.

Semantic search is computationally less intensive than BFS, as it avoids exhaustive traversal and focuses only on nodes with high semantic relevance. By prioritizing nodes that are semantically similar to the problem statement, we provide the GPT model with highly pertinent information, potentially improving proof generation quality. The disadvantages are that the effectiveness of semantic search is contingent upon the embedding model's ability to accurately capture mathematical semantics, which may be challenging for complex or abstract concepts. Important nodes that are not semantically similar based on the embedding (e.g., foundational axioms or lemmas) may be overlooked, potentially omitting crucial information required for the proof.

Regardless of the search method used, we adopted an iterative prompting strategy with the GPT model. This approach allows us to manage the amount of information provided to the GPT model, aiming to strike a balance between context richness and the model's capacity to process and utilize the information effectively.

I. Prompt / completion examples

MiniF2F example ("Prove that $x^3 + 6x$ is divisible by 3"). (a) No retrieval. Problem statement (17 tokens) + system header (68 tokens) $\rightarrow$ model says *"… we note that* $x^3 + 6x = x(x^2 + 6)$. *Because either x or x^2 is divisible by 3, their product is.* $\square$*"* — rejected by Lean (missing modular-arithmetic lemma).

(b) RAG (k=5). Adds 357 context tokens (definitions of "divides", "mod", etc.); same model now supplies a Lean-ready informal proof that auto-formalizes successfully.

(c) KG-Prover depth 3. Context grows to 1612 tokens (path via "Cubic residues mod 3"). First attempt fails, second attempt (after an error-aware re-prompt) passes Lean in 14 s wall-clock. Full prompt and intermediate completions are in our public repo.

J. Failure Scenarios

Although we see strong performance across multiple proof benchmarks, there are certain scenarios in which models & techniques fail to function optimally. Across multiple runs, we found the following possible errors:

- The informal proof is correct, but the conversion into a formal proof fails.

- The required knowledge is not in the graph, and other topics are too briefly related to be extrapolated.

In our manual analysis, we found that approximately 35% of the failures occur when the formal proof is incorrect despite the informal proof being largely valid. This suggests that the challenge often lies not in the mathematical reasoning itself, but in bridging the gap between informal and formal representations. Informal proofs frequently rely on high-level abstractions, implicit assumptions, or natural language shortcuts (e.g., “it follows that,” “by symmetry”) that do not always translate cleanly into Lean 4, which demands precision and fully explicit logic. Typical issues include omitted hypotheses, ambiguous theorem references, or imperfect formalization of induction and algebraic steps.

It is rare that traversal doesn’t gather relevant information or that the knowledge is not available and only apparent on particularly hard questions. However, for difficult questions, such as those proposed by the International Math Olympiad, the graph cannot find the most relevant nodes.